

3958 **NAME**3959 aio_cancel — cancel an asynchronous I/O request (**REALTIME**)3960 **SYNOPSIS**3961 AIO `#include <aio.h>`3962 `int aio_cancel(int fildes, struct aiocb *aiocbp);`

3963

3964 **DESCRIPTION**

3965 The *aio_cancel()* function shall attempt to cancel one or more asynchronous I/O requests
 3966 currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the
 3967 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then
 3968 all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

3969 Normal asynchronous notification shall occur for asynchronous I/O operations that are
 3970 successfully canceled. If there are requests that cannot be canceled, then the normal
 3971 asynchronous completion process shall take place for those requests when they are completed.

3972 For requested operations that are successfully canceled, the associated error status shall be set to
 3973 [ECANCELED] and the return status shall be -1. For requested operations that are not
 3974 successfully canceled, the *aiocbp* shall not be modified by *aio_cancel()*.

3975 If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which
 3976 the asynchronous operation was initiated, unspecified results occur.

3977 Which operations are cancelable is implementation-defined.

3978 **RETURN VALUE**

3979 The *aio_cancel()* function shall return the value AIO_CANCELED if the requested operation(s) 2
 3980 were canceled. The value AIO_NOTCANCELED shall be returned if at least one of the
 3981 requested operation(s) cannot be canceled because it is in progress. In this case, the state of the
 3982 other operations, if any, referenced in the call to *aio_cancel()* is not indicated by the return value
 3983 of *aio_cancel()*. The application may determine the state of affairs for these operations by using
 3984 *aio_error()*. The value AIO_ALLDONE is returned if all of the operations have already
 3985 completed. Otherwise, the function shall return -1 and set *errno* to indicate the error.

3986 **ERRORS**

3987 The *aio_cancel()* function shall fail if:

3988 [EBADF] The *fildes* argument is not a valid file descriptor.

3989 **EXAMPLES**

3990 None.

3991 **APPLICATION USAGE**

3992 The *aio_cancel()* function is part of the Asynchronous Input and Output option and need not be
 3993 available on all implementations.

3994 **RATIONALE**

3995 None.

3996 **FUTURE DIRECTIONS**

3997 None.

3998 **SEE ALSO**

3999 *aio_read()*, *aio_write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

4000 **CHANGE HISTORY**

4001 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4002 **Issue 6**

4003 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4004 implementation does not support the Asynchronous Input and Output option.

4005 The APPLICATION USAGE section is added.

4006 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/10 is applied, removing the words “to the 2
4007 calling process” in the RETURN VALUE section. The term was unnecessary and precluded 2
4008 threads. 2

4009 **NAME**4010 aio_error — retrieve errors status for an asynchronous I/O operation (**REALTIME**)4011 **SYNOPSIS**4012 AIO `#include <aio.h>`4013 `int aio_error(const struct aiocb *aiocbp);`

4014

4015 **DESCRIPTION**

4016 The *aio_error()* function shall return the error status associated with the **aiocb** structure
 4017 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the
 4018 SIO *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()*
 4019 operation. If the operation has not yet completed, then the error status shall be equal to
 4020 *[EINPROGRESS]*.

4021 **RETURN VALUE**

4022 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the
 4023 asynchronous operation has completed unsuccessfully, then the error status, as described for
 4024 SIO *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has
 4025 not yet completed, then *[EINPROGRESS]* shall be returned.

4026 **ERRORS**4027 The *aio_error()* function may fail if:

4028 *[EINVAL]* The *aiocbp* argument does not refer to an asynchronous operation whose
 4029 return status has not yet been retrieved.

4030 **EXAMPLES**

4031 None.

4032 **APPLICATION USAGE**

4033 The *aio_error()* function is part of the Asynchronous Input and Output option and need not be
 4034 available on all implementations.

4035 **RATIONALE**

4036 None.

4037 **FUTURE DIRECTIONS**

4038 None.

4039 **SEE ALSO**

4040 *aio_cancel()*, *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 4041 *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

4042 **CHANGE HISTORY**

4043 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4044 **Issue 6**

4045 The *[ENOSYS]* error condition has been removed as stubs need not be provided if an
 4046 implementation does not support the Asynchronous Input and Output option.

4047 The APPLICATION USAGE section is added.

4048 NAME

4049 aio_fsync — asynchronous file synchronization (**REALTIME**)

4050 SYNOPSIS

4051 AIO `#include <aio.h>`4052 `int aio_fsync(int op, struct aiocb *aiocbp);`

4053

4054 DESCRIPTION

4055 The *aio_fsync()* function shall asynchronously force all I/O operations associated with the file
 4056 indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp*
 4057 argument and queued at the time of the call to *aio_fsync()* to the synchronized I/O completion
 4058 state. The function call shall return when the synchronization request has been initiated or
 4059 queued to the file or device (even when the data cannot be synchronized immediately).

4060 If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to
 4061 *fdatasync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is O_SYNC,
 4062 all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as
 4063 defined for synchronized I/O file integrity completion. If the *aio_fsync()* function fails, or if the
 4064 operation queued by *aio_fsync()* fails, then, as for *fsync()* and *fdatasync()*, outstanding I/O
 4065 operations are not guaranteed to have been completed.

4066 If *aio_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to
 4067 *aio_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of
 4068 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized
 4069 fashion.

4070 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used
 4071 as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return
 4072 status, respectively, of the asynchronous operation while it is proceeding. When the request is
 4073 queued, the error status for the operation is [EINPROGRESS]. When all data has been
 4074 successfully transferred, the error status shall be reset to reflect the success or failure of the
 4075 operation. If the operation does not complete successfully, the error status for the operation shall
 4076 be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to
 4077 occur as specified in Section 2.4.1 (on page 28) when all operations have achieved synchronized
 4078 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the
 4079 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4080 completion, then the behavior is undefined.

4081 If the *aio_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to
 4082 have been successfully transferred.

4083 RETURN VALUE

4084 The *aio_fsync()* function shall return the value 0 if the I/O operation is successfully queued; 2
 4085 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

4086 ERRORS

4087 The *aio_fsync()* function shall fail if:

4088 [EAGAIN] The requested asynchronous operation was not queued due to temporary
 4089 resource limitations.

4090 [EBADF] The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument
 4091 is not a valid file descriptor open for writing.

4092 [EINVAL] This implementation does not support synchronized I/O for this file.

4093 [EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified.

4094 In the event that any of the queued I/O operations fail, *aio_fsync()* shall return the error

4095 condition defined for *read()* and *write()*. The error is returned in the error status for the

4096 asynchronous *fsync()* operation, which can be retrieved using *aio_error()*.

4097 EXAMPLES

4098 None.

4099 APPLICATION USAGE

4100 The *aio_fsync()* function is part of the Asynchronous Input and Output option and need not be

4101 available on all implementations.

4102 RATIONALE

4103 None.

4104 FUTURE DIRECTIONS

4105 None.

4106 SEE ALSO

4107 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of

4108 IEEE Std 1003.1-2001, <**aio.h**>

4109 CHANGE HISTORY

4110 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4111 Issue 6

4112 The [ENOSYS] error condition has been removed as stubs need not be provided if an

4113 implementation does not support the Asynchronous Input and Output option.

4114 The APPLICATION USAGE section is added.

4115 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/11 is applied, removing the words “to the 2

4116 calling process” in the RETURN VALUE section. The term was unnecessary and precluded 2

4117 threads. 2

4118 **NAME**4119 aio_read — asynchronous read from a file (**REALTIME**)4120 **SYNOPSIS**4121 AIO `#include <aio.h>`4122 `int aio_read(struct aiocb *aiocbp);`

4123

4124 **DESCRIPTION**

4125 The *aio_read()* function shall read *aiocbp->aio_nbytes* from the file associated with
 4126 *aiocbp->aio_fildes* into the buffer pointed to by *aiocbp->aio_buf*. The function call shall return when
 4127 the read request has been initiated or queued to the file or device (even when the data cannot be
 4128 delivered immediately).

4129 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted 2
 4130 at a priority equal to a base scheduling priority minus *aiocbp->aio_reqprio*. If Thread Execution 2
 4131 Scheduling is not supported, then the base scheduling priority is that of the calling process; 2
 4132 PIO TPS otherwise, the base scheduling priority is that of the calling thread. 2

4133 The *aiocbp* value may be used as an argument to *aio_error()* and *aio_return()* in order to
 4134 determine the error status and return status, respectively, of the asynchronous operation while it
 4135 is proceeding. If an error condition is encountered during queuing, the function call shall return
 4136 without having initiated or queued the request. The requested operation takes place at the
 4137 absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to
 4138 the operation with an *offset* equal to *aio_offset* and a *whence* equal to *SEEK_SET*. After a
 4139 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file
 4140 is unspecified.

4141 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_read()*.

4142 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 4143 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4144 completion, then the behavior is undefined.

4145 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4146 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this 2
 4147 function shall be according to the definitions of synchronized I/O data integrity completion and
 4148 synchronized I/O file integrity completion.

4149 For any system action that changes the process memory space while an asynchronous I/O is
 4150 outstanding to the address range being changed, the result of that action is undefined.

4151 For regular files, no data transfer shall occur past the offset maximum established in the open
 4152 file description associated with *aiocbp->aio_fildes*.

4153 **RETURN VALUE**

4154 The *aio_read()* function shall return the value zero if the I/O operation is successfully queued; 2
 4155 otherwise, the function shall return the value `-1` and set *errno* to indicate the error.

4156 **ERRORS**

4157 The *aio_read()* function shall fail if:

4158 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4159 resource limitations.

4160 Each of the following conditions may be detected synchronously at the time of the call to
 4161 *aio_read()*, or asynchronously. If any of the conditions below are detected synchronously, the
 4162 *aio_read()* function shall return `-1` and set *errno* to the corresponding value. If any of the

conditions below are detected asynchronously, the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to the corresponding value.

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for reading.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid, *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value.

In the case that the *aio_read()* successfully queues the I/O operation but the operation is subsequently canceled or encounters an error, the return status of the asynchronous operation is one of the values normally returned by the *read()* function call. In addition, the error status of the asynchronous operation is set to one of the error statuses normally set by the *read()* function call, or one of the following values:

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for reading.

[ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel()* request.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid.

The following condition may be detected synchronously or asynchronously:

[EOVERFLOW] The file is a regular file, *aiocbp->aio_nbytes* is greater than 0, and the starting offset in *aiocbp->aio_offset* is before the end-of-file and is at or beyond the offset maximum in the open file description associated with *aiocbp->aio_fildes*.

EXAMPLES

None.

APPLICATION USAGE

The *aio_read()* function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel(), *aio_error()*, *lio_listio()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

4201	The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:	
4202		
4203	<ul style="list-style-type: none"> • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files. 	
4204		
4205	<ul style="list-style-type: none"> • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files. 	
4206		
4207	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/12 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words “to the calling process” in the RETURN VALUE section.	2
4208		2
4209		2
4210	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/13 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of <i>aiochp</i> -> <i>aio_reqprio</i> is only required if the Prioritized Input and Output option is supported.	2
4211		2
4212		2

4213 **NAME**4214 `aio_return` — retrieve return status of an asynchronous I/O operation (**REALTIME**)4215 **SYNOPSIS**4216 AIO `#include <aio.h>`4217 `ssize_t aio_return(struct aiocb *aiocbp);`

4218

4219 **DESCRIPTION**

4220 The `aio_return()` function shall return the return status associated with the **aiocb** structure
 4221 referenced by the `aiocbp` argument. The return status for an asynchronous I/O operation is the
 4222 value that would be returned by the corresponding `read()`, `write()`, or `fsync()` function call. If the
 4223 error status for the operation is equal to `[EINPROGRESS]`, then the return status for the
 4224 operation is undefined. The `aio_return()` function may be called exactly once to retrieve the
 4225 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in
 4226 a call to `aio_return()` or `aio_error()`, an error may be returned. When the **aiocb** structure referred
 4227 to by `aiocbp` is used to submit another asynchronous operation, then `aio_return()` may be
 4228 successfully used to retrieve the return status of that operation.

4229 **RETURN VALUE**

4230 If the asynchronous I/O operation has completed, then the return status, as described for `read()`,
 4231 `write()`, and `fsync()`, shall be returned. If the asynchronous I/O operation has not yet completed,
 4232 the results of `aio_return()` are undefined.

4233 **ERRORS**4234 The `aio_return()` function may fail if:

4235 `[EINVAL]` The `aiocbp` argument does not refer to an asynchronous operation whose
 4236 return status has not yet been retrieved.

4237 **EXAMPLES**

4238 None.

4239 **APPLICATION USAGE**

4240 The `aio_return()` function is part of the Asynchronous Input and Output option and need not be
 4241 available on all implementations.

4242 **RATIONALE**

4243 None.

4244 **FUTURE DIRECTIONS**

4245 None.

4246 **SEE ALSO**

4247 `aio_cancel()`, `aio_error()`, `aio_fsync()`, `aio_read()`, `aio_write()`, `close()`, `exec`, `exit()`, `fork()`, `lio_listio()`,
 4248 `lseek()`, `read()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<aio.h>`

4249 **CHANGE HISTORY**

4250 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4251 **Issue 6**

4252 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an
 4253 implementation does not support the Asynchronous Input and Output option.

4254 The APPLICATION USAGE section is added.

4255 The `[EINVAL]` error condition is made optional. This is for consistency with the DESCRIPTION.

4256 **NAME**4257 aio_suspend — wait for an asynchronous I/O request (**REALTIME**)4258 **SYNOPSIS**

4259 AIO #include <aio.h>

```
4260       int aio_suspend(const struct aiocb * const list[], int nent,
4261                      const struct timespec *timeout);
4262
```

4263 **DESCRIPTION**

4264 The *aio_suspend()* function shall suspend the calling thread until at least one of the asynchronous
 4265 I/O operations referenced by the *list* argument has completed, until a signal interrupts the
 4266 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any
 4267 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,
 4268 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the
 4269 function shall return without suspending the calling thread. The *list* argument is an array of
 4270 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of
 4271 elements in the array. Each **aiocb** structure pointed to has been used in initiating an
 4272 asynchronous I/O request via *aio_read()*, *aio_write()*, or *lio_listio()*. This array may contain
 4273 NULL pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures
 4274 that have not been used in submitting asynchronous I/O, the effect is undefined.

4275 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of
 4276 the I/O operations referenced by *list* are completed, then *aio_suspend()* shall return with an
 4277 error. If the Monotonic Clock option is supported, the clock that shall be used to measure this
 4278 time interval shall be the CLOCK_MONOTONIC clock.

4279 **RETURN VALUE**

4280 If the *aio_suspend()* function returns after one or more asynchronous I/O operations have
 4281 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and
 4282 set *errno* to indicate the error.

4283 The application may determine which asynchronous I/O completed by scanning the associated
 4284 error and return status using *aio_error()* and *aio_return()*, respectively.

4285 **ERRORS**4286 The *aio_suspend()* function shall fail if:

4287 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the
 4288 time interval indicated by *timeout*.

4289 [EINTR] A signal interrupted the *aio_suspend()* function. Note that, since each
 4290 asynchronous I/O operation may possibly provoke a signal when it
 4291 completes, this error return may be caused by the completion of one (or more)
 4292 of the very I/O operations being awaited.

4293 **EXAMPLES**

4294 None.

4295 **APPLICATION USAGE**

4296 The *aio_suspend()* function is part of the Asynchronous Input and Output option and need not
 4297 be available on all implementations.

4298 **RATIONALE**

4299 None.

4300 **FUTURE DIRECTIONS**

4301 None.

4302 **SEE ALSO**4303 *aio_read()*, *aio_write()*, *lio_listio()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>4304 **CHANGE HISTORY**

4305 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4306 **Issue 6**4307 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4308 implementation does not support the Asynchronous Input and Output option.

4309 The APPLICATION USAGE section is added.

4310 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
4311 CLOCK_MONOTONIC clock, if supported, is used.

4312 NAME

4313 aio_write — asynchronous write to a file (**REALTIME**)

4314 SYNOPSIS

4315 AIO `#include <aio.h>`4316 `int aio_write(struct aiocb *aiocbp);`

4317

4318 DESCRIPTION

4319 The *aio_write()* function shall write *aiocbp->aio_nbytes* to the file associated with *aiocbp->aio_fildes*
 4320 from the buffer pointed to by *aiocbp->aio_buf*. The function shall return when the write request
 4321 has been initiated or, at a minimum, queued to the file or device.

4322 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted 2
 4323 at a priority equal to a base scheduling priority minus *aiocbp->aio_reqprio*. If Thread Execution 2
 4324 Scheduling is not supported, then the base scheduling priority is that of the calling process; 2
 4325 PIO TPS otherwise, the base scheduling priority is that of the calling thread. 2

4326 The *aiocbp* argument may be used as an argument to *aio_error()* and *aio_return()* in order to
 4327 determine the error status and return status, respectively, of the asynchronous operation while it
 4328 is proceeding.

4329 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 4330 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4331 completion, then the behavior is undefined.

4332 If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take
 4333 place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called
 4334 immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to
 4335 SEEK_SET. If O_APPEND is set for the file descriptor, write operations append to the file in the
 4336 same order as the calls were made. After a successful call to enqueue an asynchronous I/O
 4337 operation, the value of the file offset for the file is unspecified.

4338 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_write()*.

4339 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4340 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this 2
 4341 function shall be according to the definitions of synchronized I/O data integrity completion, and
 4342 synchronized I/O file integrity completion.

4343 For any system action that changes the process memory space while an asynchronous I/O is
 4344 outstanding to the address range being changed, the result of that action is undefined.

4345 For regular files, no data transfer shall occur past the offset maximum established in the open
 4346 file description associated with *aiocbp->aio_fildes*.

4347 RETURN VALUE

4348 The *aio_write()* function shall return the value zero if the I/O operation is successfully queued; 2
 4349 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

4350 ERRORS

4351 The *aio_write()* function shall fail if:

4352 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4353 resource limitations.

4354 Each of the following conditions may be detected synchronously at the time of the call to
 4355 *aio_write()*, or asynchronously. If any of the conditions below are detected synchronously, the

4356 *aio_write()* function shall return `-1` and set *errno* to the corresponding value. If any of the
 4357 conditions below are detected asynchronously, the return status of the asynchronous operation
 4358 shall be set to `-1`, and the error status of the asynchronous operation is set to the corresponding
 4359 value.

4360	[EBADF]	The <i>aiochp->aio_fildes</i> argument is not a valid file descriptor open for writing.	
4361	[EINVAL]	The file offset value implied by <i>aiochp->aio_offset</i> would be invalid,	2
4362 PIO		<i>aiochp->aio_reqprio</i> is not a valid value, or <i>aiochp->aio_nbytes</i> is an invalid	2
4363		value.	2

4364 In the case that the *aio_write()* successfully queues the I/O operation, the return status of the
 4365 asynchronous operation shall be one of the values normally returned by the *write()* function call.
 4366 If the operation is successfully queued but is subsequently canceled or encounters an error, the
 4367 error status for the asynchronous operation contains one of the values normally set by the
 4368 *write()* function call, or one of the following:

4369	[EBADF]	The <i>aiochp->aio_fildes</i> argument is not a valid file descriptor open for writing.
4370	[EINVAL]	The file offset value implied by <i>aiochp->aio_offset</i> would be invalid.
4371	[ECANCELED]	The requested I/O was canceled before the I/O completed due to an explicit
4372		<i>aio_cancel()</i> request.

4373 The following condition may be detected synchronously or asynchronously:

4374	[EFBIG]	The file is a regular file, <i>aiochp->aio_nbytes</i> is greater than 0, and the starting
4375		offset in <i>aiochp->aio_offset</i> is at or beyond the offset maximum in the open file
4376		description associated with <i>aiochp->aio_fildes</i> .

4377 **EXAMPLES**

4378 None.

4379 **APPLICATION USAGE**

4380 The *aio_write()* function is part of the Asynchronous Input and Output option and need not be
 4381 available on all implementations.

4382 **RATIONALE**

4383 None.

4384 **FUTURE DIRECTIONS**

4385 None.

4386 **SEE ALSO**

4387 *aio_cancel()*, *aio_error()*, *aio_read()*, *aio_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*, *lseek()*,
 4388 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

4389 **CHANGE HISTORY**

4390 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4391 Large File Summit extensions are added.

4392 **Issue 6**

4393 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 4394 implementation does not support the Asynchronous Input and Output option.

4395 The APPLICATION USAGE section is added.

4396 The following new requirements on POSIX implementations derive from alignment with the
 4397 Single UNIX Specification:

4398		
4399	• In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs	
4400	past the offset maximum established in the open file description associated with	
	<i>aiocbp->aio_fildes</i> .	
4401	• The [EFBIG] error is added as part of the large file support extensions.	
4402	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/14 is applied, rewording the	2
4403	DESCRIPTION when prioritized I/O is supported to account for threads, and removing the	2
4404	words “to the calling process” in the RETURN VALUE section.	2
4405	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/15 is applied, updating the [EINVAL]	2
4406	error, so that detection of an [EINVAL] error for an invalid value of <i>aiocbp->aio_reqprio</i> is only	2
4407	required if the Prioritized Input and Output option is supported.	2

23032 NAME

23033 lio_listio — list directed I/O (**REALTIME**)

23034 SYNOPSIS

23035 AIO #include <aio.h>

```
23036       int lio_listio(int mode, struct aiocb *restrict const list[restrict],
23037                    int nent, struct sigevent *restrict sig);
```

23038

23039 DESCRIPTION

23040 The *lio_listio()* function shall initiate a list of I/O requests with a single function call.

23041 The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in <aio.h> and
 23042 determines whether the function returns when the I/O operations have been completed, or as
 23043 soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall
 23044 wait until all I/O is complete and the *sig* argument shall be ignored.

23045 If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous
 23046 notification shall occur, according to the *sig* argument, when all the I/O operations complete. If
 23047 *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous
 23048 notification occurs as specified in Section 2.4.1 (on page 28) when all the requests in *list* have
 23049 completed.

23050 The I/O requests enumerated by *list* are submitted in an unspecified order.

23051 The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.
 23052 The array may contain NULL elements, which shall be ignored.

23053 If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list* 2
 23054 become illegal addresses before all asynchronous I/O completed and, if necessary, the 2
 23055 notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio_buf* 2
 23056 member of the **aiocb** structure pointed to by the elements of the array *list* become illegal 2
 23057 addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed, 2
 23058 the behavior is undefined. 2

23059 The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The
 23060 supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in
 23061 <aio.h>. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode*
 23062 element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read()*
 23063 with the *aio_cbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal
 23064 to LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write()* with the *aio_cbp*
 23065 equal to the address of the **aiocb** structure.

23066 The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.23067 The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.23068 The *aio_nbytes* member specifies the number of bytes of data to be transferred.

23069 The members of the **aiocb** structure further describe the I/O operation to be performed, in a
 23070 manner identical to that of the corresponding **aiocb** structure when used by the *aio_read()* and
 23071 *aio_write()* functions.

23072 The *nent* argument specifies how many elements are members of the list; that is, the length of the
 23073 array.

23074 The behavior of this function is altered according to the definitions of synchronized I/O data
 23075 integrity completion and synchronized I/O file integrity completion if synchronized I/O is
 23076 enabled on the file associated with *aio_fildes*.

23077 For regular files, no data transfer shall occur past the offset maximum established in the open
 23078 file description associated with *aiochp->aio_fildes*.

23079 If *sig->sigev_notify* is SIGEV_THREAD and *sig->sigev_notify_attributes* is a non-NULL pointer 2
 23080 and the block pointed to by this pointer becomes an illegal address prior to all asynchronous I/O 2
 23081 being completed, then the behavior is undefined. 2

23082 **RETURN VALUE**

23083 If the *mode* argument has the value LIO_NOWAIT, the *lio_listio()* function shall return the value
 23084 zero if the I/O operations are successfully queued; otherwise, the function shall return the value
 23085 -1 and set *errno* to indicate the error.

23086 If the *mode* argument has the value LIO_WAIT, the *lio_listio()* function shall return the value
 23087 zero when all the indicated I/O has completed successfully. Otherwise, *lio_listio()* shall return a
 23088 value of -1 and set *errno* to indicate the error.

23089 In either case, the return value only indicates the success or failure of the *lio_listio()* call itself,
 23090 not the status of the individual I/O requests. In some cases one or more of the I/O requests
 23091 contained in the list may fail. Failure of an individual request does not prevent completion of
 23092 any other individual request. To determine the outcome of each I/O request, the application
 23093 shall examine the error status associated with each **aiochb** control block. The error statuses so
 23094 returned are identical to those returned as the result of an *aio_read()* or *aio_write()* function.

23095 **ERRORS**

23096 The *lio_listio()* function shall fail if:

23097 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The
 23098 application may check the error status for each **aiochb** to determine the
 23099 individual request(s) that failed.

23100 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit
 23101 {AIO_MAX} to be exceeded.

23102 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than
 23103 {AIO_LISTIO_MAX}.

23104 [EINTR] A signal was delivered while waiting for all I/O requests to complete during
 23105 an LIO_WAIT operation. Note that, since each I/O operation invoked by
 23106 *lio_listio()* may possibly provoke a signal when it completes, this error return
 23107 may be caused by the completion of one (or more) of the very I/O operations
 23108 being awaited. Outstanding I/O requests are not canceled, and the application
 23109 shall examine each list element to determine whether the request was
 23110 initiated, canceled, or completed.

23111 [EIO] One or more of the individual I/O operations failed. The application may
 23112 check the error status for each **aiochb** structure to determine the individual
 23113 request(s) that failed.

23114 In addition to the errors returned by the *lio_listio()* function, if the *lio_listio()* function succeeds
 23115 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list
 23116 may have been initiated. If the *lio_listio()* function fails with an error code other than [EAGAIN],
 23117 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation
 23118 indicated by each list element can encounter errors specific to the individual read or write
 23119 function being performed. In this event, the error status for each **aiochb** control block contains the
 23120 associated error code. The error codes that can be set are the same as would be set by a *read()* or
 23121 *write()* function, with the following additional error codes possible:

23122	[EAGAIN]	The requested I/O operation was not queued due to resource limitations.
23123	[ECANCELED]	The requested I/O was canceled before the I/O completed due to an explicit <i>aio_cancel()</i> request.
23124		
23125	[EFBIG]	
23126		The <i>aioctx->aio_lio_opcode</i> is LIO_WRITE, the file is a regular file, <i>aioctx->aio_nbytes</i> is greater than 0, and the <i>aioctx->aio_offset</i> is greater than or equal to the offset maximum in the open file description associated with <i>aioctx->aio_fildes</i> .
23127		
23128		
23129	[EINPROGRESS]	
23130	[EOVERFLOW]	The <i>aioctx->aio_lio_opcode</i> is LIO_READ, the file is a regular file, <i>aioctx->aio_nbytes</i> is greater than 0, and the <i>aioctx->aio_offset</i> is before the end-of-file and is greater than or equal to the offset maximum in the open file description associated with <i>aioctx->aio_fildes</i> .
23131		
23132		
23133		

23134 EXAMPLES

23135 None.

23136 APPLICATION USAGE

23137 None.

23138 RATIONALE

23139 Although it may appear that there are inconsistencies in the specified circumstances for error
 23140 codes, the [EIO] error condition applies when any circumstance relating to an individual
 23141 operation makes that operation fail. This might be due to a badly formulated request (for
 23142 example, the *aio_lio_opcode* field is invalid, and *aio_error()* returns [EINVAL]) or might arise from
 23143 application behavior (for example, the file descriptor is closed before the operation is initiated,
 23144 and *aio_error()* returns [EBADF]).

23145 The limitation on the set of error codes returned when operations from the list shall have been
 23146 initiated enables applications to know when operations have been started and whether
 23147 *aio_error()* is valid for a specific operation.

23148 FUTURE DIRECTIONS

23149 None.

23150 SEE ALSO

23151 *aio_read()*, *aio_write()*, *aio_error()*, *aio_return()*, *aio_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
 23152 *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

23153 CHANGE HISTORY

23154 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

23155 Large File Summit extensions are added.

23156 Issue 6

23157 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 23158 implementation does not support the Asynchronous Input and Output option.

23159 The *lio_listio()* function is marked as part of the Asynchronous Input and Output option.

23160 The following new requirements on POSIX implementations derive from alignment with the
 23161 Single UNIX Specification:

- 23162 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
 23163 past the offset maximum established in the open file description associated with
 23164 *aioctx->aio_fildes*. This change is to support large files.

23165	• The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support large	
23166	files.	
23167	The DESCRIPTION is updated to avoid use of the term “must” for application requirements.	
23168	The restrict keyword is added to the <i>lio_listio()</i> prototype for alignment with the	
23169	ISO/IEC 9899:1999 standard.	
23170	Issue 6	2
23171	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/53 is applied, adding new text for	2
23172	symmetry with the <i>aio_read()</i> and <i>aio_write()</i> functions to the DESCRIPTION.	2
23173	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/54 is applied, adding text to the	2
23174	DESCRIPTION making it explicit that the user is required to keep the structure pointed to by	2
23175	<i>sig->sigev_notify_attributes</i> valid until the last asynchronous operation finished and the	2
23176	notification has been sent.	2